11-22-00

A

jc915 U.S. PTO 11/20/00

jc930 U.S. PTO 09/717187 11/20/00

UTILITY PATENT APPLICATION TRANSMITTAL

(New Nonprovisional Applications Under 37 CFR § 1.53(b))

Attorney Docket No. 50277-0378

TO THE COMMISSIONER FOR PATENTS:

Transmitted herewith is the patent application of ( ) application identifier or (X) first named inventor, Vikram Joshi, et al., entitled METHOD AND APPARATUS FOR DEBUGGING A SOFTWARE PROGRAM USING DYNAMIC DEBUG PATHCES AND COPY ON WRITE VIEWS, for a(n):

(X) Original Patent Application.

( ) Continuing Application (prior application not abandoned):
- ( ) Continuation ( ) Divisional ( ) Continuation-in-part (CIP) of prior application No: _____ Filed on: _____.
- ( ) A statement claiming priority under 35 USC § 120 has been added to the specification.

Enclosed are:
- (X) Specification 26 Total Pages; (X) Drawing(s) 3 Total Sheets; (X) Cover Sheet 1 Page
- (X) Oath or Declaration: 3 Pages
  - (X) A Newly Executed Combined Declaration and Power of Attorney:
    - (X) Signed. ( ) Unsigned. ( ) Partially Signed.
  - ( ) A Copy from a Prior Application for Continuation/Divisional (37 CFR § 1.63(d)).
    - ( ) Incorporation by Reference. The entire disclosure of the prior application, from which a copy of the oath or declaration is supplied, is considered as being part of the disclosure of the accompanying application and is hereby incorporated herein by reference.
    - ( ) Signed Statement Deleting Inventor(s) Named in the Prior Application. (37 CFR § 163(d)(2)).
- (X) Power of Attorney (2 pgs) (X) Return Receipt Postcard.
- ( ) Associate Power of Attorney. ( ) A Check in the amount of $_____ for the Filing Fee.
- ( ) Preliminary Amendment. ( ) Information Disclosure Statement and Form PTO-1449.
- (X) A Duplicate Copy of this Form for Processing Fee Against Deposit Account.
- ( ) A Certified Copy of Priority Documents (if foreign priority is claimed).
- ( ) Applicant claims small entity status. See 37 CFR 1.27.
- ( ) Statement(s) of Status as a Small Entity Filed in Prior Application, Status Still Proper and Desired.
- (X) Other: Assignment Recordation Cover Sheet and executed Assignment (5 pgs)

CLAIMS AS FILED				
FOR	NO. FILED	NO. EXTRA	RATE	FEE
Total Claims	16	0	$18.00	$ 0.00
Independent Claims	2	0	$80.00	$ 0.00
Multiple Dependent Claims (if applicable)				$0.00
Assignment Recording Fee				$40.00
Basic Filing Fee				$710.00
			Total Filing Fee	$750.00

Charge $ 750.00 to Oracle Deposit Account 150635 pursuant to 37 CFR § 1.25. At any time during the pendency of this application, please charge any fees required or credit any overpayment to the Oracle Deposit Account.

Respectfully submitted,

By: Carina M. Tan

Carina M. Tan, Reg. No. 45,769

Date: November 20, 2000

Correspondence Address:

Hickman Palermo Truong & Becker, LLP
1600 Willow Street
San Jose, California 95125-5106
Telephone: (408) 414-1080
Facsimile: (408) 414-1076

I hereby certify that this is being deposited with the U.S. Postal Service "Express Mail Post Office to Addressee" service under 37 CFR § 1.10 on the date indicated below and is addressed to:

Commissioner for Patents
Box Patent Application
Washington, D.C. 20231

By: Casey Moore

Typed Name: Casey Moore

Express Mail Label No.: EL652871070US

Date of Deposit: November 20, 2000

PTO/SB/17 (12/99)
Approved for use through 09/30/2000. OMB 0651-0032
Patent and Trademark Office: U.S. DEPARTMENT OF COMMERCE
Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it displays a valid OMB control number.

FEE TRANSMITTAL for FY 2000

Patent fees are subject to annual revision,
Small Entity payments must be supported by a small entity statement, otherwise large entity fees must be paid. See Forms PTO/SB/09-12.
See 37 C.F.R. §§ 1.27 AND 1.28

TOTAL AMOUNT OF PAYMENT	($) 750.00

Complete if Known	
Application Number	NYA
Filing Date	November 20, 2000
First Named Inventor	Vikram Joshi, et al.
Examiner Name	NYA
Group/Art Unit	NYA
Attorney Docket No.	50277-0378

METHOD OF PAYMENT (check one)

1. ☒ The Commissioner is hereby authorized to charge indicated fees and credit any overpayments to:

Deposit Account Number: 150635

Deposit Account Name: Oracle Corporation

☒ Charge Any Additional Fee Required Under 37 CFR §§ 1.16 and 1.17

2. ☐ Payment Enclosed:

☐ Check ☐ Money Order ☐ Other

3. ☐ Applicant claims small entity status. See 37 CFR 1.27

FEE CALCULATION

1. BASIC FILING FEE

Large Entity Fee Code	Large Entity Fee ($)	Small Entity Fee Code	Small Entity Fee ($)	Fee Description	Fee Paid
101	710	201	355	Utility filing fee	710.00
106	320	206	160	Design filing fee	
107	490	207	245	Plant filing fee	
108	710	208	355	Reissue filing fee	
114	150	214	75	Provisional filing fee	

SUBTOTAL (1) ($) 710.00

2. EXTRA CLAIM FEES

			Extra Claims		Fee from Below		Fee Paid
Total Claims	16	-20**=	0	X	18.00	=	0.00
Independent Claims	2	- 3**=	0	X	80.00	=	0.00
Multiple Dependent						=	

**or number previously paid, if greater; For Reissues, see below

Large Entity Fee Code	Large Entity Fee ($)	Small Entity Fee Code	Small Entity Fee ($)	Fee Description
103	18	203	9	Claims in excess of 20
102	80	202	40	Independent claims in excess of 3
104	270	204	135	Multiple dependent claim, if not paid
109	80	209	40	**Reissue independent claims over original patent
110	18	210	9	**Reissue claims in excess of 20 and over original patent

SUBTOTAL (2) ($) 0.00

FEE CALCULATION (continued)

3. ADDITIONAL FEES

Large Entity Fee Code	Large Entity Fee ($)	Small Entity Fee Code	Small Entity Fee ($)	Fee Description	Fee Paid
105	130	205	65	**Surcharge – late filing fee or oath**	
127	50	227	25	Surcharge – late provisional filing fee or cover sheet.	
139	130	139	130	Non-English specification	
112	920*	112	920*	Requesting publication of SIR prior to Examiner action	
113	1,840*	113	1,840*	Requesting publication of SIR after Examiner action	
115	110	215	55	Extension for reply within first month	
116	390	216	195	Extension for reply within second month	
117	890	217	445	Extension for reply within third month	
118	1,390	218	695	Extension for reply within fourth month	
128	1,890	228	945	Extension for reply within fifth month	
119	310	219	155	Notice of Appeal	
120	310	220	155	Filing a brief in support of an appeal	
121	270	221	135	Request for oral hearing	
138	1,510	138	1,510	Petition to institute a public use proceeding	
140	110	240	55	Petition to revive – unavoidable	
141	1,210	241	605	Petition to revive – unintentional	
142	1,210	242	605	Utility issue fee (or reissue)	
143	430	243	215	Design issue fee	
144	580	244	290	Plant issue fee	
122	130	122	130	Petitions to the Commissioner	
123	50	123	50	Petitions related to provisional applications	
126	240	126	240	Submission of information Disclosure Stmt	
581	40	581	40	Recording each patent assignment per property (times number of properties)	40.00
146	690	246	345	Filing a submission after final rejection (37 CFR § 1.129(a))	
149	690	249	345	For each additional invention to be examined (37 CFR § 1.129(b))	

Other fee (specify) _____

Other fee (specify) _____

*Reduced by Basic Filing Fee Paid SUBTOTAL (3) ($) 40.00

SUBMITTED BY				Complete (if applicable)	
Name (Print/Type)	Carina M. Tan	Registration No. (Attorney/Agent)	45,769	Telephone	(408) 414-1080
Signature	Carina M. Tan			Date	November 20, 2000

WARNING: **Information on this form may become public. Credit card information should not be included on this form. Provide credit card information and authorization on PTO-2038.**

Burden Hour Statement. This form is estimated to take 0.2 hours to complete Time will vary depending upon the needs of the individual case. Any comments on the amount of time you are required to complete this form should be sent to the Chief Information Officer, Patent and Trademark Office, Washington, DC 20231. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. SEND TO: Box Patent Applicatin, Commissioner for Patents, Washington, DC 20231.

OID 1999-173-02

50277-0378 *Patent*

UNITED STATES PATENT APPLICATION

FOR

METHOD AND APPARATUS FOR DEBUGGING A SOFTWARE PROGRAM USING DYNAMIC DEBUG PATCHES AND COPY ON WRITE VIEWS

INVENTORS:

VIKRAM JOSHI
ALEX TSUKERMAN
SHARI YAMAGUCHI

PREPARED BY:

HICKMAN PALERMO TRUONG & BECKER, LLP
1600 WILLOW STREET
SAN JOSE, CA 95125-5106
(408) 414-1080

METHOD AND APPARATUS FOR DEBUGGING A SOFTWARE PROGRAM USING DYNAMIC DEBUG PATCHES AND COPY ON WRITE VIEWS

CROSS REFERENCE TO RELATED APPLICATIONS

This application is related to and claims domestic priority under 35 U.S.C. § 119(e) from prior U.S. Provisional Patent Application Serial Number 60/166,598 filed on November 19, 1999 entitled "Debugging Techniques And Fast SGA Dumps For Deferred Analysis Of The Database", by inventors Vikram Joshi, Alex Tsukerman, and Shari Yamaguchi, the entire disclosure of which is hereby incorporated by reference as if fully set forth herein.

This application is related to U.S. Patent Application Serial Number 09/649,310 filed on August 28, 2000, entitled "Method And Apparatus For Debugging A Software Program", by inventors Vikram Joshi, Alex Tsukerman, and Shari Yamaguchi, the entire disclosure of which is hereby incorporated by reference as if fully set forth herein.

This application is related to U.S. Patent Application Serial Number __/___,___ filed on the same day herewith entitled "Fast Database State Dumps To File For Deferred Analysis Of A Database", by inventors Vikram Joshi, Alex Tsukerman, and Shari Yamaguchi (Attorney docket number 50277-379, OID 1999-173-03), the entire disclosure of which is hereby incorporated by reference as if fully set forth herein.

This application is related to U.S. Patent Application Serial Number __/___,___ filed on the same day herewith entitled "A Debug And Data Collection Mechanism Utilizing A Difference In Database State By Using Consecutive Snapshots Of A Database State", by inventors Vikram Joshi, Alex Tsukerman, and Shari Yamaguchi (Attorney docket number 50277-380, OID 1999-173-04), the entire disclosure of which is hereby incorporated by reference as if fully set forth herein.

FIELD OF THE INVENTION

The present invention generally relates to debugging software programs and, more specifically, to techniques for debugging database systems.

BACKGROUND OF THE INVENTION

In a database system, an area of system memory is allocated and one or more processes are started to execute one or more transactions. The database server communicates with connected user processes and performs tasks on behalf of the user. These tasks typically include the execution of transactions. The combination of the allocated system memory and the processes executing transactions is commonly termed a database "server" or "instance".

Like most software systems, a database server has complicated shared memory structures. A shared memory structure contains data and control information for a portion of a database system. Because of software, hardware, or firmware bugs that may exist in a complex database system, shared memory structures may become logically incorrect. When structures become logically incorrect, the database system is likely to fail. Database failure is typically discovered in the following ways: by checking consistency of structures; by verifying certain assumptions; or by running into corrupted pointers. Attempting to process corrupted pointers will lead to a "crash," after which normal database operation is no longer possible.

A major responsibility of the database administrator is to be prepared for the possibility of hardware, software, network, process, or system failure. When shared structures are presumed to be corrupted, the best course of action for a database administrator is to cease further processing of the database. If a failure occurs such that the operation of a database system is affected, the administrator must usually recover the database and return the database to normal operations as quickly as possible. Recovery should protect the database and associated users from unnecessary problems and avoid or reduce the possibility of having to duplicate work manually.

Recovery operations vary depending on the type of failure that occurred, the structures affected, and the type of recovery that is performed. If no files are lost or damaged, recovery may amount to no more than rebooting the database system. On the other hand, if data has been lost, recovery requires additional steps in order to put the database back into normal working order.

Once the database is recovered or rebooted, the immediate problem is quickly resolved, but because the root cause is still undetermined and therefore unresolved, the error condition may resurface, potentially causing several additional outages. Therefore, it is still important to diagnose the state of the structures and data surrounding the database failure. Such a diagnosis may provide valuable information that can reduce the chance of failure in the future. As a practical matter, diagnosing the failure may lead to determining which vendor's hardware or software is responsible for the database failure. Such information is valuable for a vendor's peace of mind, if nothing else. Thus, competing with the goal of recovering the database as quickly as possible, is the goal of determining why the database system failed in the first place.

Unfortunately, even with traditional techniques of diagnosing a database failure, the system administrator is usually unable to obtain a sufficient amount of clues to determine why the failure happened. A deliberate and thorough diagnosis of the failure may require an unacceptable amount of database downtime. For example, any amount of downtime over 30 minutes may be extremely costly for a database that is associated with a highly active web site. Too much downtime may have unduly expensive business ramifications, such as lost revenue and damage to the reputation of the web site owner.

Another problem with traditional debugging techniques is that they can be intrusive. For example, a database system that supports the Structured Query Language (SQL) may be debugged by compiling SQL statements and running them against the database. The act of compiling and executing the SQL statements changes the state of

the database system. Thus, the mere act of diagnosing the problem can easily cloud the problem or make the problem worse because diagnosis may involve altering the state of the database. Diagnosing the problem typically involves using debugging software, which calls for peeking and poking into data structures within the complex memory structures of the database systems. Although the data structures are best left untouched upon a failure, diagnosing the failure may involve working directly on the same data structures from which data is to be obtained. Nevertheless, it is important to preserve the original data and not change the data from its state at time of failure. A customer of the database may take issue to changing the database, because such a change may jeopardize or even destabilize their database system.

Effective diagnosis, however, requires getting as much information as possible out of the data structures. It may be useful here to refer to Heisenberg's uncertainty principle, which effectively states that the closer an object is analyzed, the more the object materially changes because the mere act of analyzing is intrusive. Applying this principle to the act of diagnosing a database failure, a typical debugging operation is naturally intrusive. Thus, it is difficult to be non-intrusive on a database and at the same time obtain a sufficient amount of meaningful data for debugging.

Traditional debugging techniques involve formatting certain parts of the database system and displaying this formatted portion in a human-readable form. This human-readable form can be set aside for later analysis, for example, after the database has been recovered or is no longer down. The entire memory of the database server is not dumped because an average database server is very large, typically between about 200 megabytes and about 100 gigabytes of unformatted binary and data. On the portion of the database that is formatted, an educated guess is made of the key data structures that are potential causes of the problem.

Unfortunately, such a debugging technique provides diagnosis only to the

database server's end-memory state, which is the state after the database has been shutdown. Because the end-memory state is being analyzed separately from the database, the programmer performing the debugging does not have access to the real database and some of the database's persistent structures. Some of these persistent structures could be on disk or, in a multiple node system, on other nodes. For example, in a parallel server configuration, the persistent structures needed for debugging could reside on other servers. Thus, the technique of separately debugging portions of the database prevents the programmer from using the data that can only be obtained from the database itself.

Further, where debug operations are performed on the database while the database is down, multiple programmers cannot each privately diagnose the failure. Rather, the key data structures are typically diagnosed by having one programmer in front of a console inputting debug commands, while other programmers gather around issuing advice. Multiple programmers individually debugging the database is unadvisable using existing debugging techniques because the act of inputting debug commands is intrusive, as mentioned above. Each programmer's work would interfere with the concurrent debugging progress of fellow programmers.

For the foregoing reasons, what is needed is a method of debugging a software program, such as a database system, that is non-intrusive, yet allows for a comprehensive assessment of a failure.

SUMMARY OF THE INVENTION

Techniques are provided for allowing multiple persons to concurrently test software patches on a software program or debug a problem of the software program. Each person preferably has their own private view, which consists of (1) copied portions of the software program that reflect modifications made by that person, and (2) the portions of the preserved software program that the person has not modified. Providing a private view to each person allows each person to test and debug privately, independently, and concurrently with others. Each private view may be extensively explored and modified without affecting the memory state of the software program that existed at the time the software program was shutdown.

Accordingly, at any time, each private view may be refreshed to the state of the software program that existed at the time of shutdown. Faster diagnosis of the problem may therefore be accomplished because a debugger does not have to peek cautiously and slowly into the inner-workings of the software program. Similarly, the testing of various potential solutions to bugs in the program may be accomplished efficiently without affecting the memory state of the software program that existed at the time the software program was shutdown. Thus, where downtime of a software program must be kept to a minimum, the present techniques allow for performing quick and comprehensive diagnostics and testing of potential solutions to problems in the software program.

BRIEF DESCRIPTION OF THE DRAWINGS

The present invention is illustrated by way of example, and not by way of limitation, in the figures of the accompanying drawings and in which like reference numerals refer to similar elements and in which:

FIG. 1 illustrates how data is copied to preserve the memory state of a software program before being modified in response to a technique for debugging and testing of potential solutions for a software program;

FIG. 2 is a flowchart of a technique that allows for non-intrusive debugging and testing of potential solutions for a software program; and

FIG. 3 is a block diagram that illustrates a computer system upon which an embodiment of the invention may be implemented.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

Techniques for non-intrusive testing of potential solutions for and debugging of a software program are described. In the following description, for the purposes of explanation, numerous specific details are set forth in order to provide a thorough understanding of the present invention. It will be apparent, however, to one skilled in the art that the present invention may be practiced without these specific details. In other instances, well-known structures and devices are shown in block diagram form in order to avoid unnecessarily obscuring the present invention.

TESTING AND DIAGNOSTIC TECHNIQUE

A database administrator may cause a database system to cease execution for a number of reasons, which are discussed above. Diagnosing a database will typically lead to modifying data in the database while it is down. As explained above, it is desirable to preserve the memory state that existed at the time of failure or at the time the database was shutdown.

FIG. 1 illustrates how, using the techniques described herein, data is copied to preserve the memory state of a software program before being modified. For the purpose of explanation, it will be assumed that the software program is a database server. However, the present techniques are not limited to any particular type of program. A database administrator, for example, "freezes" a portion of a database to preserve the memory state of the database system. Preserving the database may include suspending a failed process within the database system. Various techniques may be used to freeze the state of a database server. One such technique is described in U.S. Patent Application No. 09/223,660 entitled "METHOD AND SYSTEM FOR DIAGNOSTIC PRESERVATION OF THE STATE OF A COMPUTER SYSTEM" filed by Wei Hu and Juan Loaiza on December 30, 1998, the contents of which is incorporated herein by reference.

The act of preserving the database may be initiated by giving the software

program an explicit "freeze" user command. Alternatively, the act of preserving the database may be initiated in response to an automatic trigger that fires when an error event is detected. The techniques described hereafter indicate how debugging and testing may be performed, even by multiple users concurrently, without changing any data in the preserved portion 102. Any operation that could cause any change within the preserved portion is disabled with respect to the preserved portion 102.

In one embodiment, the software program is a database server that is composed of a memory portion referred to as a "database instance", and a set of data on disk referred to as "datafiles". In addition, one database server may be shared in a hardware cluster with additional database instances residing on different nodes, but still sharing access to the same set of datafiles. Preferably, the database administrator will be able to issue a command to preserve only the database instance that has failed, thereby detaching the preserved database instance from the cluster membership. In such a situation, it is important to exercise care while using the preserved database instance in this state, and to not affect the integrity of the datafiles or other database instances. Preferably, a database administrator only uses such a detached preserved database instance for debugging and testing operations. Detaching a database instance from a cluster involves isolating the instance from the rest of the cluster using software and/or hardware means.

SECONDARY SOFTWARE PROGRAMS

While the software program is preserved, debugging and or testing operations may begin. Debugging or testing may involve a second software program. The second software program may be a software patch that fixes a bug in the software program. Techniques are provided to allow for testing of the software patch without compromising the memory state of the original software program. Alternatively, the second software program may be a diagnostic tool for debugging the software program. Any such second

software program is herein generally referred to as a "secondary" software program. FIG. 1 shows the preserved portion 102 having segments of data. These segments of data are typically pages of memory. Assume that a user has written a secondary software program that is either a diagnostic tool or a potential patch the software program. The user may then compile and dynamically link the secondary software program to the original software program. Many operating systems provide for the compilation and dynamically linkage of separate software programs.

Assume that when the user executes the secondary software program, segments of the preserved portion 102 are accessed. For example, execution of the secondary results in a read operation that accesses data 104. The execution of the second software program may also call for a write operation to be performed on some data within the preserved portion 102. The data within the reserved portion that is targeted by the write operation is referred to herein as targeted data 106a. In response to an attempt to perform a write operation on data within the targeted data 106a, a copy is made of the targeted data 106a. The actual modification that would have been made to the targeted data 106a is instead made to the copy, creating a modified copy 106b. In one embodiment, the modified copy 106b is a copy-on-write page of memory.

In subsequent operations, relative to the execution of the software program, the modified copy 106b takes the place of the targeted data 106a. Thus, if the execution of the secondary software program involves a subsequent read operation of targeted data 106a, the read operation would be performed on the modified copy 106b. Similarly, if the execution of the secondary software program would involve further modification to the targeted data 106a, the modification would once again be performed on the modified copy 106b.

For simplification purposes, FIG. 1 shows a scenario in which the target portion 206a includes only one segment. Therefore, a modified copy 106b has been made of

only one target portion 106a. However, the execution of the secondary software program may actually involve modifications to many areas of a software program, and therefore cause the generation of modified copies of a multitude of segments.

In one embodiment, the memory segments of the preserved portion 102 are pages in memory. Preferably, a page map is used to keep track of all the pages of the modified copies for each user. For example, the modified copy 106b may be a copy-on-write page, the address of which is kept in a page map. Using this copying technique, the original preserved portion 102 of the software program is unaltered. The page mapping software and/or hardware ensures that, for the user that caused creation of the modified page, the modified page is mapped at the same virtual address as the original page, thus preserving the integrity of data structure references, indices, and pointers. The modified page is a modified copy of the original page, and the user that caused the creation of the modified page has a private view of the modified page. Such modified pages are herein referred to as "view private" modified pages.

The modified copies for a user may be discarded at anytime. Thus, a fresh testing and debug session may be initiated at any time using the preserved portion 102 and the aforementioned copying technique.

MAINTAINING PER-USER MODIFICATION DATA

According to one embodiment, a separate set of modified copies are maintained for each user, based on the modifications made by that user. Specifically, each user sees (1) the modified copies that have been generated in response to the execution of secondary software programs executed by that user, and (2) the preserved portions of the software program that have not been modified in response to the execution of secondary software programs initiated by that user. The modified copies may be managed in a view private fashion using any one of a number of techniques, including page mapping software and hardware techniques.

Consequently, multiple programmers may debug and test potential solutions to problems in the software program concurrently, independently, and privately. The debug and testing progress of one programmer will not affect the debug and testing progress of another programmer. Accordingly, any number of debug and testing sessions may be generated and later destroyed. Such multiple-session debugging and testing should lead to a relatively quick and comprehensive assessment of the failure and testing of various potential solutions to the failure. For example, in FIG. 1, in response to compiling, dynamically linking and executing software program 1, which is a secondary software program, a user 1 modifies 106a and 106b is created. In response to compiling, dynamically linking and executing software program 2, which is another secondary software program that is distinct from software program 1, a user 2 modifies 106a and 106c is created. User 1 is unable to see the changes in 106c, and user 2 is unable to see the changes in 106b.

SHARING SECONDARY PROGRAMS

In addition, if user 1 wishes to share software program 1, user 1 may provide access to software program 1 by publishing a symbolic name associated with software program 1 in slot 108 of FIG. 1. Multiple users may concurrently, independently, and privately execute software program 1 by using the corresponding symbolic name that is published in the preserved portion of the software program. For example, in response to user 3 concurrently, independently, and privately executing software program 1, 106d in FIG. 1 is created. Because modified copies may be managed in a view private fashion, user 3 is unable to see the changes in either 106b or 106c.

MULTIPLE SECONDARY PROGRAMS IN A SINGLE SESSION

One debug and testing session can be used to execute various secondary software programs that are separate from the original software program, which is being debugged.

The programmer of a particular debug and testing session has a private view of the original software program, which is being debugged, in a privately modified state. This privately modified state is a side-effect of the aforementioned copying technique. In the case of a secondary software program that is a diagnostic tool, extensive exploration of the privately viewed data has the benefit of extracting valuable state that may point to the problem. In other words, the privately viewed data may be extensively explored and modified without fear of altering the preserved portion 102, which represents the original data. In the case of a secondary software program that is a potential solution to a problem in the original software program, testing of the potential solution may be safely performed before publishing and or permanently adopting the potential solution.

Where the software program is a database system, the debugging and testing may be performed, at least in part, by executing various secondary software programs that have been separately compiled and dynamically linked to the software program that is being debugged. The secondary software programs are executed using existing shared database state (*i.e.*, the preserved portion 102) and the aforementioned copy-on-write technique. Additionally, the secondary software programs may be shared with multiple concurrent users by publishing, in the preserved portion, a corresponding symbolic name associated with the secondary software program.

FIG. 1 is an illustration of the privately viewed data of a debug and testing session at a particular point in time. Thus, at a particular point in time, a programmer, such as user 1, sees the preserved portion 102, minus the targeted data 106a, plus the modified copy 106b.

PRESERVING PERSISTENT DATA

In another embodiment, the software program is a database system that has persistent structures, such as persistent database tables. Persistent structures could be on disk or, in a multiple node system, reside in database instances on other nodes. For

example, in a parallel server configuration, the persistent structures needed for debugging could reside on other servers. The execution of a secondary software program may call for accessing data, such as a persistent structure, that is outside the current database instance being debugged. For such data that resides outside the database instance being debugged, it is preferable to mount the data in a read-only mode into the database prior to performing debug and testing operations. Such a mounting step facilitates reads from outside data, such as persistent database tables. Debug and testing operations cannot write to the read-only data, and therefore will not make changes to the original persistent structures. Further, when any operation attempts to write to the read-only data, the debug and testing system preferably produces a logical error message, which the debug and testing system makes known to the user.

After copying outside data into the database in a read-only mode, outside data may be treated as part of the preserved portion 102. That is, a user is allowed to perform operations that modify the data, but those operations cause the creation of separate modified copies, and leave the original data intact. Thus, an embodiment of the present invention is applicable to debugging a database and testing of potential solutions to problems in the database where execution of secondary software programs call for accessing data that is outside the current database instance.

THE DEBUGGING OPERATION

FIG. 2 is a flowchart of a debug and testing technique that allows for non-intrusive debugging of a software program as well as testing potential solutions to problems in the software program. At block 202, a memory state of a preserved portion of the software program is preserved. As mentioned in the discussion with reference to FIG. 1, preserving the memory state may include suspending an application that has failed. At block 204, a secondary software program is compiled and dynamically linked to the original software program. At block 206, the secondary software program is

executed causing the creation of a copy of targeted data 106a, if the execution of the secondary software program would normally cause modification to targeted data 106a in the portion of the software program that is being preserved.

HARDWARE OVERVIEW

FIG. 3 is a block diagram that illustrates a computer system 300 upon which an embodiment of the invention may be implemented. Computer system 300 includes a bus 302 or other communication mechanism for communicating information, and a processor 304 coupled with bus 302 for processing information. Computer system 300 also includes a main memory 306, such as a random access memory (RAM) or other dynamic storage device, coupled to bus 302 for storing information and instructions to be executed by processor 304. Main memory 306 also may be used for storing temporary variables or other intermediate information during execution of instructions to be executed by processor 304. Computer system 300 further includes a read only memory (ROM) 308 or other static storage device coupled to bus 302 for storing static information and instructions for processor 304. A storage device 310, such as a magnetic disk or optical disk, is provided and coupled to bus 302 for storing information and instructions.

Computer system 300 may be coupled via bus 302 to a display 312, such as a cathode ray tube (CRT), for displaying information to a computer user. An input device 314, including alphanumeric and other keys, is coupled to bus 302 for communicating information and command selections to processor 304. Another type of user input device is cursor control 316, such as a mouse, a trackball, or cursor direction keys for communicating direction information and command selections to processor 304 and for controlling cursor movement on display 312. This input device typically has two degrees of freedom in two

axes, a first axis (e.g., x) and a second axis (e.g., y), that allows the device to specify positions in a plane.

The invention is related to the use of computer system 300 for implementing the techniques described herein. According to one embodiment of the invention, those techniques are implemented by computer system 300 in response to processor 304 executing one or more sequences of one or more instructions contained in main memory 306. Such instructions may be read into main memory 306 from another computer-readable medium, such as storage device 310. Execution of the sequences of instructions contained in main memory 306 causes processor 304 to perform the process steps described herein. In alternative embodiments, hard-wired circuitry may be used in place of or in combination with software instructions to implement the invention. Thus, embodiments of the invention are not limited to any specific combination of hardware circuitry and software.

The term "computer-readable medium" as used herein refers to any medium that participates in providing instructions to processor 304 for execution. Such a medium may take many forms, including but not limited to, non-volatile media, volatile media, and transmission media. Non-volatile media includes, for example, optical or magnetic disks, such as storage device 310. Volatile media includes dynamic memory, such as main memory 306. Transmission media includes coaxial cables, copper wire and fiber optics, including the wires that comprise bus 302. Transmission media can also take the form of acoustic or light waves, such as those generated during radio-wave and infra-red data communications.

Common forms of computer-readable media include, for example, a floppy disk, a flexible disk, hard disk, magnetic tape, or any other magnetic medium, a CD-ROM, any other optical medium, punchcards, papertape, any other physical medium with patterns of holes, a

RAM, a PROM, and EPROM, a FLASH-EPROM, any other memory chip or cartridge, a carrier wave as described hereinafter, or any other medium from which a computer can read.

Various forms of computer readable media may be involved in carrying one or more sequences of one or more instructions to processor 304 for execution. For example, the instructions may initially be carried on a magnetic disk of a remote computer. The remote computer can load the instructions into its dynamic memory and send the instructions over a telephone line using a modem. A modem local to computer system 300 can receive the data on the telephone line and use an infra-red transmitter to convert the data to an infra-red signal. An infra-red detector can receive the data carried in the infra-red signal and appropriate circuitry can place the data on bus 302. Bus 302 carries the data to main memory 306, from which processor 304 retrieves and executes the instructions. The instructions received by main memory 306 may optionally be stored on storage device 310 either before or after execution by processor 304.

Computer system 300 also includes a communication interface 318 coupled to bus 302. Communication interface 318 provides a two-way data communication coupling to a network link 320 that is connected to a local network 322. For example, communication interface 318 may be an integrated services digital network (ISDN) card or a modem to provide a data communication connection to a corresponding type of telephone line. As another example, communication interface 318 may be a local area network (LAN) card to provide a data communication connection to a compatible LAN. Wireless links may also be implemented. In any such implementation, communication interface 318 sends and receives electrical, electromagnetic or optical signals that carry digital data streams representing various types of information.

Network link 320 typically provides data communication through one or more networks to other data devices. For example, network link 320 may provide a connection through local network 322 to a host computer 324 or to data equipment operated by an Internet Service Provider (ISP) 326. ISP 326 in turn provides data communication services through the world wide packet data communication network now commonly referred to as the "Internet" 328. Local network 322 and Internet 328 both use electrical, electromagnetic or optical signals that carry digital data streams. The signals through the various networks and the signals on network link 320 and through communication interface 318, which carry the digital data to and from computer system 300, are exemplary forms of carrier waves transporting the information.

Computer system 300 can send messages and receive data, including program code, through the network(s), network link 320 and communication interface 318. In the Internet example, a server 330 might transmit a requested code for an application program through Internet 328, ISP 326, local network 322 and communication interface 318. In accordance with the invention, one such downloaded application implements the techniques described herein.

The received code may be executed by processor 304 as it is received, and/or stored in storage device 310, or other non-volatile storage for later execution. In this manner, computer system 300 may obtain application code in the form of a carrier wave.

CONCLUSION

Techniques are described above for debugging a software program and for testing potential solutions to problems in the software program. In a preferred embodiment, the software program is preserved before debug and testing operations are performed on the software program. A secondary software program, which may either be a diagnostic tool

or a potential solution to problems in the software program, is compiled and dynamically linked to the software program. A copy-on-write step is performed on accessed data that is to be modified as a result of executing the secondary software program. During the debug and testing process, modifications are made to the copied data of the software program, and not to the preserved portion 102. The secondary software program may be shared with other concurrent users of the software program by publishing, in the preserved portion 102, a corresponding symbolic name associated with the secondary software program. While the above description provides a database system as an example of a software program, the present invention generally applies to all software programs.

In the foregoing specification, the invention has been described with reference to specific embodiments thereof. It will, however, be evident that various modifications and changes may be made thereto without departing from the broader spirit and scope of the invention. The specification and drawings are, accordingly, to be regarded in an illustrative rather than a restrictive sense.

CLAIMS

What is claimed is:

1. A method of debugging a first software program, the method comprising the steps of:
preserving a memory state of a preserved portion of the first software program;
dynamically linking a second software program to the first software program without deallocating from volatile memory the first software program;
executing the second software program; and
when execution of the second software program would otherwise cause modification to targeted data that is in the preserved portion of the first software program, making a copy of the targeted and modifying the copy to generate a modified copy of the targeted data without modifying the targeted data that is in the preserved portion of the first software program.

2. The method of Claim 1, further comprising the steps of:
publishing in the preserved portion of the first software program a corresponding symbolic name associated with the second software program; and
multiple users accessing the second software program is accessed through the corresponding symbolic name.

3. The method of Claim 1, wherein the first software program is a database system.

4. The method of Claim 1, wherein the step of preserving a memory state further

includes the step of suspending a failed application of the database system.

5. The method of Claim 1, further including the step of, in response to a subsequent attempt to access the targeted data in the preserved portion of the first software program, accessing the modified copy of the targeted data.

6. The method of Claim 5, wherein the steps of dynamically linking and executing are initiated by a particular user, and wherein the step of accessing the modified copy occurs only if that particular user initiates the subsequent attempt to access the targeted data.

7. The method of Claim 1, wherein:
the steps of dynamically linking and executing the second software program are performed by a first user;
the modified copy is a first modified copy of the targeted data; and
the method further comprises the steps of:
after the first modified copy has been created for the first user, a second user executing performing an operation which, when executed, would cause modification to the targeted data in the preserved portion; and
performing the operation by making a second copy of the targeted data and modifying the second copy to generate a second modified copy of the targeted data, the second modified copy being separate from the first modified copy and from the preserved portion.

8. The method of Claim 7, further comprising the steps of:

after the first and second modified copies have been created for the first user and second user respectively, a third user dynamically linking and executing a third software program which, when executed, would cause modification to the targeted data in the preserved portion; and

making a third copy of the targeted data and modifying the third copy to generate a third modified copy, the third modified copy being separate from the first modified copy, from the second modified copy, and from the preserved portion.

9. A computer-readable medium bearing instructions for debugging a first software program, the instructions arranged, when executed by one or more processors, to cause the one or more processors to perform the steps of:

preserving a memory state of a preserved portion of the first software program;

dynamically linking a second software program to the first software program without deallocating from volatile memory the first software program;

executing the second software program; and

when execution of the second software program would otherwise cause modification to targeted data that is in the preserved portion of the first software program, making a copy of the targeted and modifying the copy to generate a modified

copy of the targeted data without modifying the targeted data that is in the preserved portion of the first software program.

10. The computer-readable medium of Claim 9, further comprising the steps of:

publishing in the preserved portion of the first software program a corresponding symbolic name associated with the second software program; and

multiple users accessing the second software program is accessed through the corresponding symbolic name.

11. The computer-readable medium of Claim 9, wherein the first software program is a database system.

12. The computer-readable medium of Claim 9, wherein the step of preserving a memory state further includes the step of suspending a failed application of the database system.

13. The computer-readable medium of Claim 9, further including the step of, in response to a subsequent attempt to access the targeted data in the preserved portion of the first software program, accessing the modified copy of the targeted data.

14. The computer-readable medium of Claim 13, wherein the steps of dynamically

linking and executing are initiated by a particular user, and wherein the step of accessing the modified copy occurs only if that particular user initiates the subsequent attempt to access the targeted data.

15. The computer-readable medium of Claim 9, wherein:
the steps of dynamically linking and executing the second software program are performed by a first user;
the modified copy is a first modified copy of the targeted data; and
the method further comprises the steps of:
after the first modified copy has been created for the first user, a second user executing performing an operation which, when executed, would cause modification to the targeted data in the preserved portion; and
performing the operation by making a second copy of the targeted data and modifying the second copy to generate a second modified copy of the targeted data, the second modified copy being separate from the first modified copy and from the preserved portion.

16. The computer-readable medium of Claim 15, further comprising the steps of:
after the first and second modified copies have been created for the first user and second user respectively, a third user dynamically linking and executing a third software program which, when executed, would cause modification to the targeted data in the preserved portion; and

making a third copy of the targeted data and modifying the third copy to generate a third modified copy, the third modified copy being separate from the first modified copy, from the second modified copy, and from the preserved portion.

ABSTRACT OF THE DISCLOSURE

A method and apparatus for debugging a software program is provided that is non-intrusive and allows multiple persons to debug concurrently in view private sessions. In one example, a method includes preserving a memory state of a portion of a software program, such as a database system. A second software program is compiled and dynamically linked, and which when executed, would normally cause modification to targeted data in the preserved portion of the software program. The second software program is executed by making a copy of the targeted data in the preserved portion of the software program. The copy is modified to generate a modified copy of the targeted data without modifying the data that is in the preserved portion of the software program. In subsequent accesses, the user that issued that executed the second software program accesses the modified copy whenever the user would have otherwise accessed the corresponding preserved portion. The second software program is made accessible to other users of the database system by publishing in the preserved portion a corresponding symbolic name associated with the second software program. If another user accesses the second software program and executes it, then another copy of the targeted data is made for that user. As before the copy is modified to generate a modified copy of the targeted data without modifying the data that is in the preserved portion of the software program.

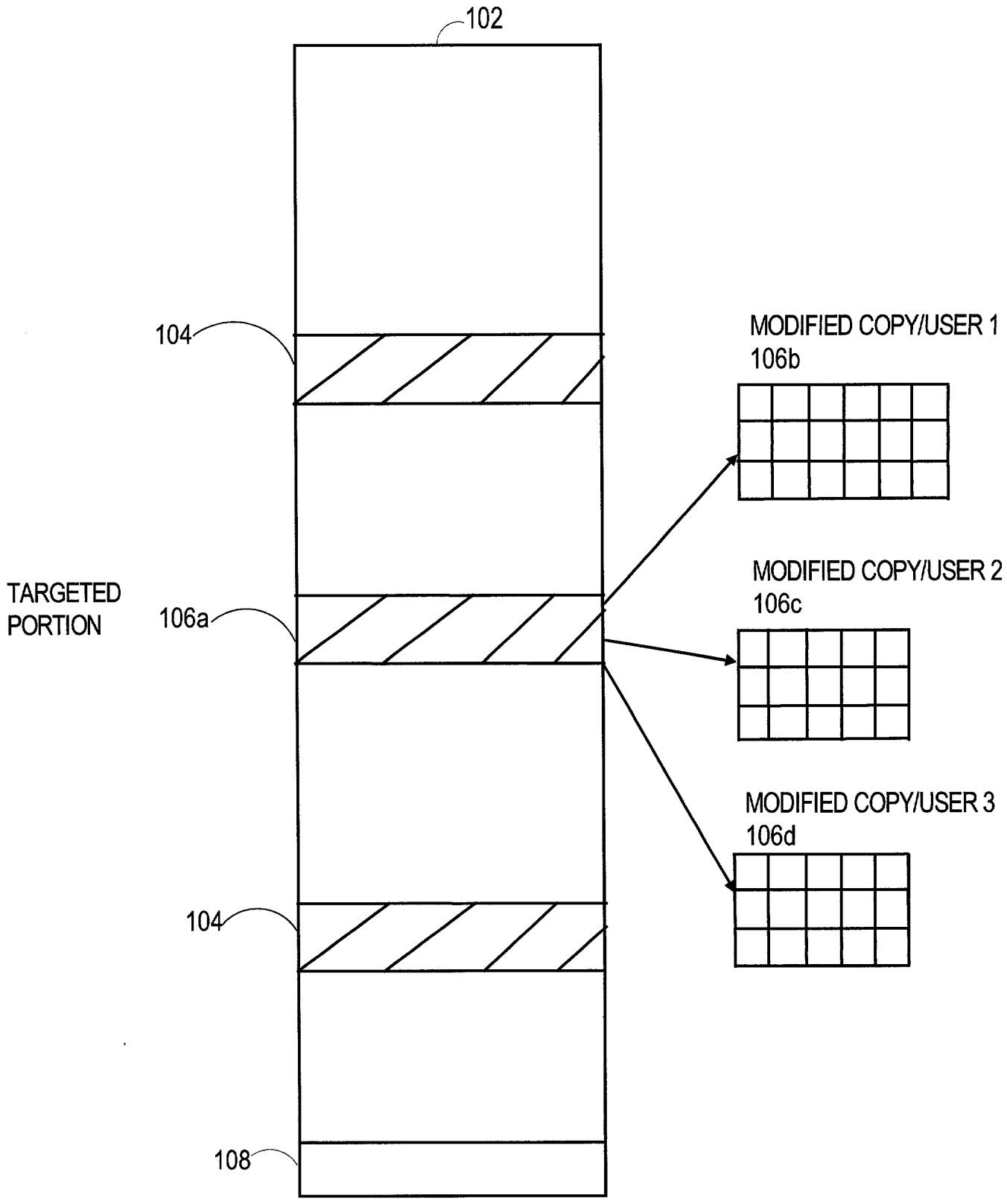

102
104
MODIFIED COPY/USER 1
106b
MODIFIED COPY/USER 2
106c
TARGETED
PORTION
106a
MODIFIED COPY/USER 3
106d
104
108

FIG. 1

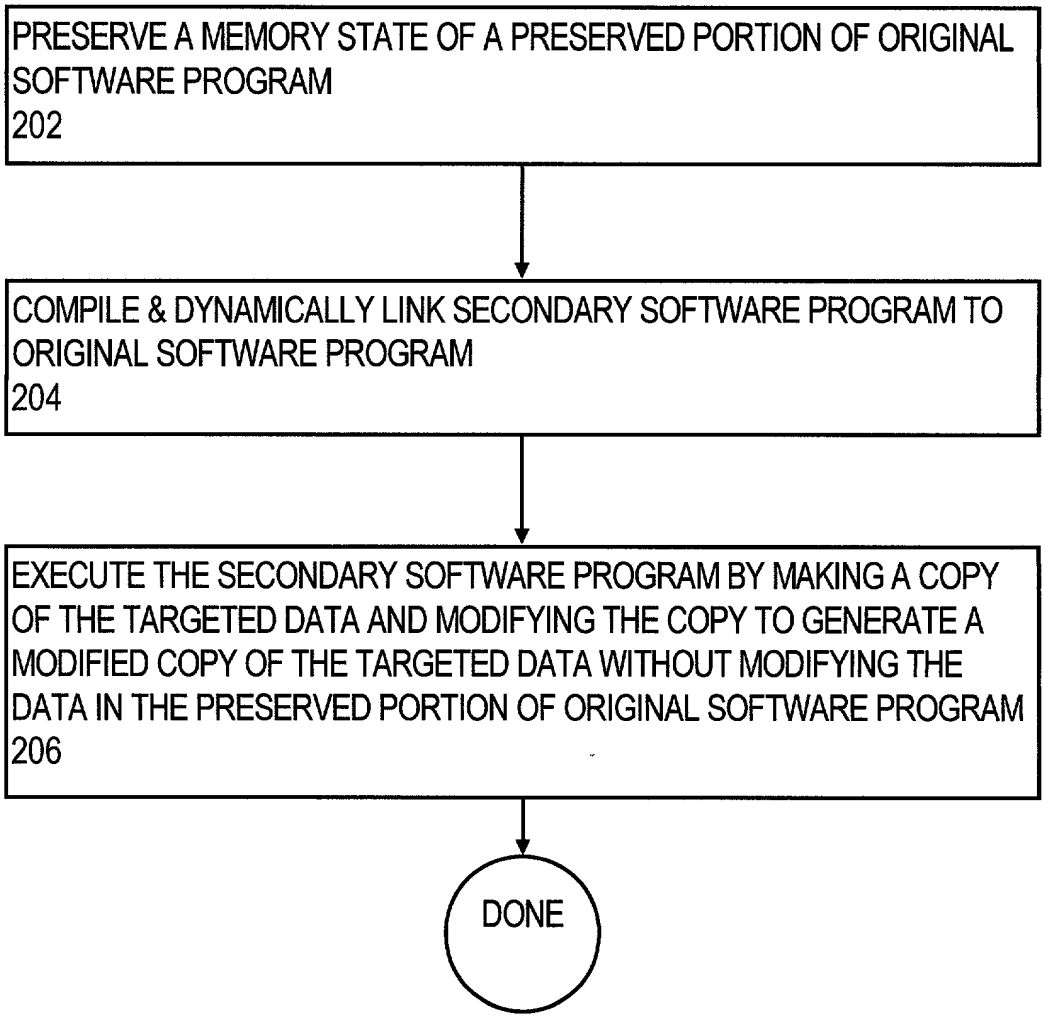
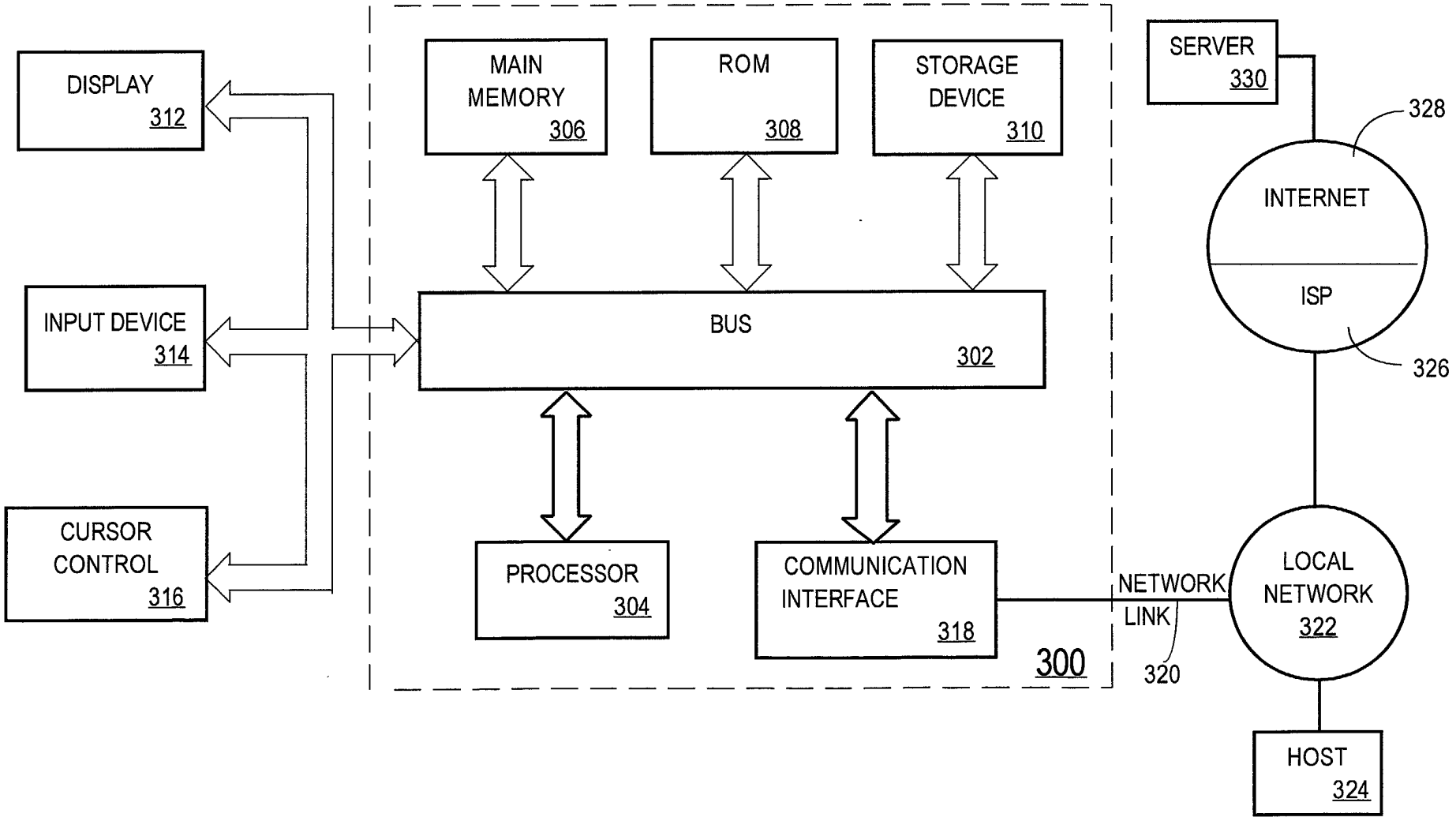

PRESERVE A MEMORY STATE OF A PRESERVED PORTION OF ORIGINAL SOFTWARE PROGRAM
202
COMPILE & DYNAMICALLY LINK SECONDARY SOFTWARE PROGRAM TO ORIGINAL SOFTWARE PROGRAM
204
EXECUTE THE SECONDARY SOFTWARE PROGRAM BY MAKING A COPY OF THE TARGETED DATA AND MODIFYING THE COPY TO GENERATE A MODIFIED COPY OF THE TARGETED DATA WITHOUT MODIFYING THE DATA IN THE PRESERVED PORTION OF ORIGINAL SOFTWARE PROGRAM
206
DONE

FIG. 2

FIG. 3



Attorney's Docket No.: 50277-0378 Patent

DECLARATION FOR PATENT APPLICATION

As a below named inventor, I hereby declare that:

My residence, post office address and citizenship are as stated below, next to my name.

I believe I am an original, first, and joint inventor of the subject matter which is claimed and for which a patent is sought on the invention entitled

"METHOD AND APPARATUS FOR DEBUGGING A SOFTWARE PROGRAM USING DYNAMIC DEBUG PATCHES AND COPY ON WRITE VIEWS"

the specification of which

_X_ is attached hereto.
___ was filed on _____ as
United States Application Number _____,
or PCT International Application Number _____,
and was amended on _____.
(if applicable)

I hereby state that I have reviewed and understand the contents of the above-identified specification, including the claims(s), as amended by any amendment referred to above.

I acknowledge the duty to disclose all information known to me to be material to patentability as defined in Title 37, Code of Federal Regulations, Section 1.56 (copy attached).

I hereby claim foreign priority benefits under Title 35, United States Code, Section 119(a)-(d), on any foreign application(s) for patent or inventor's certificate listed below and have also identified below any foreign application for patent or inventor's certificate having a filing date before that of the application on which priority is claimed:

Prior Foreign Application(s)			Priority Claimed	
(Number)	(Country)	(Day/Month/Year Filed)	Yes	No
(Number)	(Country)	(Day/Month/Year Filed)	Yes	No
(Number)	(Country)	(Day/Month/Year Filed)	Yes	No

I hereby claim the benefit under Title 35, United States Code, Section 119(e) of any United States provisional application(s) listed below

(Application Number) (Filing Date)

(Application Number) (Filing Date)

I hereby claim benefit under Title 35, United States Code, Section 120 of any United States application(s) listed below and, insofar as the subject matter of each of the claims of this application is not disclosed in the prior United States application in the manner provided by the first paragraph of Title 35, United States Code, Section 112, I acknowledge the duty to disclose all information known to me to be material to patentability as defined in Title 37, Code of Federal Regulations, Section 1.56 (copy attached) which became available between the filing date of the prior application and the national or PCT International filing date of this application:

(Application Number)	(Filing Date)	(Status - patented, pending, abandoned)
(Application Number)	(Filing Date)	(Status - patented, pending, abandoned)
(Application Number)	(Filing Date)	(Status - patented, pending, abandoned)

I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code and that such willful false statements may jeopardize the validity of the application or any patent issued thereon.

Full Name of Sole/First Inventor (given name, family name) VIKRAM JOSHI

Inventor's Signature Vikram Joshi Date 11/16/00

Residence 18552 Favre Ridge Rd., Los Gatos, California 95033 Citizenship USA
(City, State) (Country)

Post Office Address _____

Full Name of Second Inventor (given name, family name) ALEX TSUKERMAN

Inventor's Signature _____ Date 11/16/00

Residence 30 Port Royal, Foster City, California 94404 Citizenship USA
(City, State) (Country)

Post Office Address _____

Full Name of Third Inventor (given name, family name) SHARI YAMAGUCHI

Inventor's Signature _____ Date 11/16/00

Residence 4951 Formby Ct., San Jose, California 95138 Citizenship USA
(City, State) (Country)

Post Office Address _____

Title 37, Code of Federal Regulations, Section 1.56
Duty to Disclose Information Material to Patentability

(a) A patent by its very nature is affected with a public interest. The public interest is best served, and the most effective patent examination occurs when, at the time an application is being examined, the Office is aware of and evaluates the teachings of all information material to patentability. Each individual associated with the filing and prosecution of a patent application has a duty of candor and good faith in dealing with the Office, which includes a duty to disclose to the Office all information known to that individual to be material to patentability as defined in this section. The duty to disclose information exists with respect to each pending claim until the claim is canceled or withdrawn from consideration, or the application becomes abandoned. Information material to the patentability of a claim that is canceled or withdrawn from consideration need not be submitted if the information is not material to the patentability of any claim remaining under consideration in the application. There is no duty to submit information which is not material to the patentability of any existing claim. The duty to disclose all information known to be material to patentability is deemed to be satisfied if all information known to be material to patentability of any claim issued in a patent was cited by the Office or submitted to the Office in the manner prescribed by §§ 1.97(b)-(d) and 1.98. However, no patent will be granted on an application in connection with which fraud on the Office was practiced or attempted or the duty of disclosure was violated through bad faith or intentional misconduct. The Office encourages applicants to carefully examine:

(1) Prior art cited in search reports of a foreign patent office in a counterpart application, and

(2) The closest information over which individuals associated with the filing or prosecution of a patent application believe any pending claim patentably defines, to make sure that any material information contained therein is disclosed to the Office.

(b) Under this section, information is material to patentability when it is not cumulative to information already of record or being made of record in the application, and

(1) It establishes, by itself or in combination with other information, a prima facie case of unpatentability of a claim; or

(2) It refutes, or is inconsistent with, a position the applicant takes in:

(i) Opposing an argument of unpatentability relied on by the Office, or

(ii) Asserting an argument of patentability.

A prima facie case of unpatentability is established when the information compels a conclusion that a claim is unpatentable under the preponderance of evidence, burden-of-proof standard, giving each term in the claim its broadest reasonable construction consistent with the specification, and before any consideration is given to evidence which may be submitted in an attempt to establish a contrary conclusion of patentability.

(c) Individuals associated with the filing or prosecution of a patent application within the meaning of this section are:

(1) Each inventor named in the application;

(2) Each attorney or agent who prepares or prosecutes the application; and

(3) Every other person who is substantively involved in the preparation or prosecution of the application and who is associated with the inventor, with the assignee or with anyone to whom there is an obligation to assign the application.

(d) Individuals other than the attorney, agent or inventor may comply with this section by disclosing information to the attorney, agent, or inventor.

Docket No. 50277-0378

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re application of: Group Art Unit No.: NYA

Vikram Joshi, et al. Examiner: Not Yet Assigned

Serial No.: NYA

Filed on: Concurrently herewith

For: METHOD AND APPARATUS FOR DEBUGGING A SOFTWARE PROGRAM USING DYNAMIC DEBUG PATCHES AND COPY ON WRITE VIEWS

POWER OF ATTORNEY
AND REVOCATION OF PREVIOUS POWERS

Commissioner for Patents
Washington, D.C. 20231

Sir:

Oracle Corporation, a Delaware corporation having a place of business at 500 Oracle Parkway, Box 5OP7, Redwood Shores, California 94065, certifies that to the best of its knowledge and belief it is the assignee or is entitled to ownership of the entire right, title, and interest in and to the above-referenced patent application by virtue of an Assignment filed concurrently herewith and represents that the undersigned is a representative authorized and empowered to sign on behalf of Oracle Corporation, which hereby revokes all powers of attorney previously given and appoints the following attorney(s) and/or agent(s): Edward A. Becker, Reg. No. 37,777; Marcel K. Bingham, Reg. No. 42,327; Carl L. Brandt, Reg. No. 44,555; Brian D. Hickman, Reg. No. 35,894; Craig G. Holmes, Reg. No. 44,770; Christopher J. Palermo, Reg. No. 42,056; Carina M. Tan, Reg. No. 45,769; and Bobby K. Truong, Reg. No. 37,499 all of

HICKMAN PALERMO TRUONG & BECKER LLP
1600 Willow Street
San Jose, California 95125-5106

and

Sanjay Prasad, Reg. No. 36,247; and Roger Kennedy, Reg. No. 44,823, of ORACLE CORPORATION

with full power of substitution and revocation, to prosecute this application and to transact all business in the Patent and Trademark Office connected therewith. Send all future correspondence to

the attention of Brian D. Hickman, Reg. No. 35,894, care of the above address and direct all telephone calls to the same at (408) 414-1080.

Assignee of Interest:
Oracle Corporation

Dated: 11/16/00

By: [signature]
Name: Satyam Prasad
Title: Chief Patent Counsel

Address of Assignee of Interest:
Oracle Corporation
500 Oracle Parkway - Box 659507
Redwood Shores, CA 94065

Respectfully submitted,
HICKMAN PALERMO TRUONG & BECKER LLP

Dated: 11/20/00

By: [signature]
Name: Brian D. Hickman
Reg. No.: 35,894

1600 Willow Street
San Jose, California 95125-5106
Telephone: (408) 414-1080
Facsimile: (408) 414-1076